# Memory-Augmented Monte Carlo Tree Search

**Chenjun Xiao, Jincheng Mei and Martin Müller**
Computing Science, University of Alberta
Edmonton, Canada
{chenjun,jmei2,mmueller}@ualberta.ca

## Abstract

This paper proposes and evaluates Memory-Augmented Monte Carlo Tree Search (M-MCTS), which provides a new approach to exploit generalization in online real-time search. The key idea of M-MCTS is to incorporate MCTS with a memory structure, where each entry contains information of a particular state. This memory is used to generate an approximate value estimation by combining the estimations of similar states. We show that the memory based value approximation is better than the vanilla Monte Carlo estimation with high probability under mild conditions. We evaluate M-MCTS in the game of Go. Experimental results show that M-MCTS outperforms the original MCTS with the same number of simulations.

## Introduction

The key idea of Monte Carlo Tree Search (MCTS) is to construct a search tree of states evaluated by fast Monte Carlo simulations (Coulom 2006). Starting from a given game state, many thousands of games are simulated by randomized self-play until an outcome is observed. The state value is then estimated as the mean outcome of the simulations. Meanwhile, a search tree is maintained to guide the direction of simulation, for which bandit algorithms can be employed to balance exploration and exploitation (Kocsis and Szepesvári 2006). However, with large state spaces, the accuracy of value estimation cannot be effectively guaranteed, since the mean value estimation is likely to have high variance under relatively limited search time. Inaccurate estimation can mislead building the search tree and severely degrade the performance of the program.

Recently, several machine learning approaches have been proposed to deal with this drawback of MCTS. For example, deep neural networks are employed to learn domain knowledge and approximate a state value function. They are integrated with MCTS to provide heuristics which can improve the search sample efficiency in practice (Silver et al. 2016; Tian and Zhu 2015).

The successes of the machine learning methods can be mostly contributed to the power of *generalization*, *i.e.*, similar states share information. Generalized domain knowledge is usually represented by function approximation, such as a deep neural network, which is trained *offline* from an expert move dataset or self-generated simulations (Silver et al. 2016).

Compared with the amount of research done on discovering generalization from an offline learning procedure, not too much attention has focused on exploiting the benefits of generalization during the online real-time search. The current paper proposes and evaluates a Memory-Augmented MCTS algorithm to provide an alternative approach that takes advantage of *online* generalization. We design a *memory*, where each entry contains information about a particular state, as the basis to construct an online value approximation. We demonstrate that this memory-based framework is useful for improving the performance of MCTS in both theory and practice, using an experiment in the game of Go.

The remainder of the paper is organized as follows: After preliminaries introduced in Section 2, we theoretically analyze the memory framework in Section 3. The proposed Memory-Augmented MCTS algorithm is presented in Section 4. Related work and experimental results are shown in Section 5 and 6, respectively. In Section 7, we come to our conclusion and future work.

## Preliminaries

### The Setting

Let $\mathcal{S}$ be the set of all possible states of a search problem. For $s \in \mathcal{S}$, let $\hat{V}(s) = \frac{1}{N_s}\sum_{t=1}^{N_s} R_{s,t}$ denote the value estimation of state $s$ from simulations, where $R_{s,t}$ is the outcome of a simulation, and $N_s$ is the number of simulations starting from state $s$. The true value of a state $s$ is denoted by $V^*(s)$. The main idea of our Memory-Augmented MCTS algorithm is to approximate value estimations with the help of a memory, each

entry of which contains the feature representation and simulation statistics of a particular state. The approximate value estimation is performed as follows: given a memory $\mathcal{M}$ and a state $x$, we find the $M$ most similar states $\mathcal{M}_x \subset \mathcal{M}$ according to a distance metric $d(\cdot, x)$, and compute a memory-based value estimation $\hat{V}_{\mathcal{M}}(x) = \sum_{i=1}^{M} w_i(x)\hat{V}(i), s.t. \sum_{i=1}^{M} w_i(x) = 1$.

Let $X_{s,t} = |R_{s,t} - V^*(s)|$ be the sampling error from the $t$th simulation of state $s$. In the analysis of the most popular MCTS algorithm UCT (Kocsis and Szepesvári 2006), $X_{s,t}$ is assumed to be $\sigma^2$-subgaussian, so the sampling error has zero mean and its variance is upper bounded by $\sigma^2$ (Boucheron, Lugosi, and Massart 2013). We also adopt the same assumption in our analysis. We denote the value estimation error of state $s$ by $\delta_s = |\hat{V}(s) - V^*(s)|$, and the true value difference between states $s$ and $x$ by $\varepsilon_{s,x} = |V^*(s) - V^*(x)|$. Using the property of subgaussian variables, $\delta_s$ is $\frac{\sigma^2}{N_s}$-subgaussian (Boucheron, Lugosi, and Massart 2013). Let $\varepsilon_M = \max_{i \in \mathcal{M}_x} \varepsilon_{i,x}$, we assume that our memory addressing scheme is able to control $\varepsilon_M$ within the range $[0, \varepsilon]$. The following lemma states the concentration property of subgaussian variables.

**Lemma 1.** *(Boucheron, Lugosi, and Massart 2013) If $X$ is $\sigma^2$-subgaussian, then $P(X \geq \epsilon) \leq \exp(-\frac{\epsilon^2}{2\sigma^2})$.*

### Monte Carlo Tree Search

MCTS builds a tree to evaluate states with fast simulations (Coulom 2006). Each node in the tree corresponds to a specific state $s \in \mathcal{S}$, and contains simulation statistics $\hat{V}(s)$ and $N(s)$. At each iteration of the algorithm, one simulation starts from an initial state $s_0$, and proceeds in two stages: in-tree and rollout. When a state $s_t$ is already represented in the current search tree, a *tree policy* is used to select an action to go to the next state. The most popular choice of the tree policy is to use bandit algorithms such as UCB1 (Kocsis and Szepesvári 2006). For states outside the tree, a *roll-out policy* is used to simulate a game until the end, where a trajectory of visited states $\mathcal{T} = \{s_0, s_1, \ldots, s_T\}$ and a final return $R$ are obtained. The statistics of $s \in \mathcal{T}$ in the tree are updated according to:

$$N(s) \leftarrow N(s) + 1$$
$$\hat{V}(s) \leftarrow \hat{V}(s) + \frac{R - \hat{V}(s)}{N(s)}$$

In addition the search tree is grown. In the simplest scheme, the first visited node that is not yet in the tree is added to it.

### Entropy Regularized Policy Optimization

We denote the probability simplex by $\Delta = \{\mathbf{w} : \mathbf{w} \geq 0, \mathbf{1} \cdot \mathbf{w} = 1\}$, and denote the entropy function by $H(\mathbf{w}) = -\mathbf{w} \cdot \log \mathbf{w}$. For any vector $\mathbf{q} \in \mathbb{R}^n$, the entropy-regularized optimization problem is to find the solution of

$$\max_{\mathbf{w} \in \Delta} \{\mathbf{w} \cdot \mathbf{q} + \tau H(\mathbf{w})\} \tag{1}$$

where $\tau > 0$ is the *temperature* parameter. This problem has recently drawn much attention in the reinforcement learning community (Nachum et al. 2017; Haarnoja et al. 2017; Ziebart 2010). One nice property of this problem is that given the vector $\mathbf{q}$, it has a closed form solution. Define the scalar value function $F_\tau$ (the "softmax") by $F_\tau(\mathbf{q}) = \tau \log(\sum_{i=1}^{M} e^{q_i/\tau})$, and the vector-valued function $f_\tau(\mathbf{q})$ (the "soft indmax") by $f_\tau(\mathbf{q}) = \frac{e^{\mathbf{q}/\tau}}{\sum_{i=1}^{M} e^{q_i/\tau}} = e^{(\mathbf{q} - F_\tau(\mathbf{q}))/\tau}$, where the exponentiation is component-wise. Note that $f_\tau$ maps any real valued vector into a probability distribution. The next lemma states the connection between $F_\tau$, $f_\tau$ and the entropy regularized optimization problem.

**Lemma 2.** *(Nachum et al. 2017; Haarnoja et al. 2017; Ziebart 2010)*

$$\begin{aligned} F_\tau(\mathbf{q}) =& \max_{\mathbf{w} \in \Delta} \{\mathbf{w} \cdot \mathbf{q} + \tau H(\mathbf{w})\} \\ =& f_\tau(\mathbf{q}) \cdot \mathbf{q} + \tau H(f_\tau(\mathbf{q})) \end{aligned}$$

## Value Approximation with Memory

In our approach, the memory is used to provide an approximate value function $\hat{V}_{\mathcal{M}}(x) = \sum_{i=1}^{M} w_i(x)\hat{V}(i)$, where $\sum_{i=1}^{M} w_i(x) = 1$ are the weights and $M$ is a parameter defining the neighbouring states in the memory according to some distance metric $d(\cdot, x)$. One question naturally arises, is this memory-based value approximation better than the vanilla mean outcome estimation? In this section we attempt to answer this question by showing that $|\hat{V}_{\mathcal{M}}(x) - V^*(x)| \leq \delta_x$ for state $x$ with high probability under a mild condition. We first show a trivial bound for $\Pr(|\hat{V}_{\mathcal{M}}(x) - V^*(x)| \leq \delta_x)$, then provide an improved bound with entropy regularized policy.

### A Trivial Probability Bound

The first step is to upper bound $|\hat{V}_{\mathcal{M}}(x) - V^*(x)|$ using the triangle inequality:

$$\begin{aligned} & |\sum_{i=1}^{M} w_i(x)\hat{V}(i) - V^*(x)| \\ \leq & \sum_{i=1}^{M} w_i(x)|\hat{V}(i) - V^*(x)| \\ \leq & \sum_{i=1}^{M} w_i(x)(|\hat{V}(i) - V^*(i)| + |V^*(i) - V^*(x)|) \\ = & \sum_{i=1}^{M} w_i(x)(\delta_i + \varepsilon_{i,x}) \end{aligned} \tag{2}$$

Let $\delta_M = \max_{i \in \mathcal{M}_x} \delta_i$ and $\varepsilon_M = \max_{i \in \mathcal{M}_x} \varepsilon_{i,x}$. Using the fact that $\sum_{i=1}^{M} w_i(x) = 1$, we can further take an upper bound of (2) by $\sum_{i=1}^{M} w_i(x)(\delta_i + \varepsilon_{i,x}) \leq \delta_M + \varepsilon_M$. This upper bound is very loose, since we do not specify any particular choice of the weights $\mathbf{w}$. With a standard probability argument we can immediately get the following:

**Theorem 1.** *For states $x$ satisfying $\alpha_x = \delta_x - \varepsilon > 0$, let $n_{min} = min_{i \in \mathcal{M}_x} N_i$. Then with probability at least $1 - \beta$, our memory-based value function approximation has less error than $\delta_x$ provided that:*

$$n_{min} \geq \frac{2\sigma^2}{\alpha_x^2} \log(M/\beta) \tag{3}$$

Condition (3), under which the high probability bound can be guaranteed, is quite severe. It requires that the minimum simulation numbers of all addressed memory entries are sufficiently large. This trivial bound is weak since the upper bound (2) depends on the worst memory entry addressed, without specifying any choice of the weights $\mathbf{w}$. We show that the entropy regularized policy optimization can help us to fix this problem.

### Improved Probability Bound with Entropy Regularized Policy

We now provide an improvement of the previous bound by specifying the choice of the weights $\mathbf{w}$ using entropy regularized optimization. Let $\mathbf{c}$ be a vector where $c_i = \delta_i + \varepsilon_{i,x}, 1 \leq i \leq M$. Our choice of $\mathbf{w}$ should minimize the upper bound (2), which is equivalent to:

$$\max_{\mathbf{w} \in \Delta} \{\mathbf{w} \cdot (-\mathbf{c})\} \tag{4}$$

This linear optimization problem has solution $w_j = 1$ for $j = \text{argmin}_i(\delta_i + \varepsilon_{i,x})$ and $w_k = 0$ for $k \neq j$. However, in practice we do not know the accurate value of $\delta_i$ and $\varepsilon_{i,x}$ and applying this deterministic policy may cause the problem of addressing the wrong entries. We provide an approximation by solving the entropy regularized version of this optimization problem:

$$\max_{\mathbf{w} \in \Delta} \{\mathbf{w} \cdot (-\mathbf{c}) + \tau H(\mathbf{w})\} \tag{5}$$

As $\tau$ approaches zero, we recover the original problem (4). According to **Lemma 2**, the closed form solution of problem (5) is $F_\tau(-\mathbf{c}) = \tau \log(\sum_{i=1}^{M} e^{-c_i/\tau})$ by setting $\mathbf{w} = f_\tau(-\mathbf{c})$. By equation (2), $-f_\tau(-\mathbf{c}) \cdot (-\mathbf{c}) = -F_\tau(-\mathbf{c}) + \tau H(f_\tau(-\mathbf{c})) \leq -F_\tau(-\mathbf{c}) + \tau \log M$. Therefore, to show $\Pr\{(2) \leq \delta\} \geq 1 - \beta$ for some small constant $\beta$, it suffices to show that $\Pr\{-F_\tau(-\mathbf{c}) + \tau \log M \leq \delta\} \geq 1 - \beta$.

**Theorem 2.** *For states $x$ satisfying $\alpha_x = \delta_x - \varepsilon > 0$, let $n = \sum_{i=1}^{M} N_i$. By choosing the weight $\mathbf{w} = f_\tau(-\mathbf{c}) = e^{-\mathbf{c}/\tau} / \sum_{i=1}^{M} e^{-c_i/\tau}$, with probability at least $1 - \beta$ our memory-based value function approximation has less error than $\delta_x$ provided that:*

$$n \geq \frac{2\sigma^2}{(\alpha_x - \tau \log M)^2} \log(1/\beta) \tag{6}$$

*Proof.* We show that under condition (6), it can be guaranteed that $\Pr(-F_\tau(-\mathbf{c}) + \tau \log M \leq \delta_x) \geq 1 - \beta$.

$$\begin{aligned}
&\Pr\left(-\tau \log(\sum_{i=1}^{M} \exp(-c_i/\tau)) \leq \delta_x - \tau \log M\right) \\
&= \Pr\left(\sum_{i=1}^{M} \exp(-c_i/\tau) \geq \exp(-(\delta_x - \tau \log M)/\tau)\right) \\
&\geq \Pr\left(\sum_{i=1}^{M} \exp(-\delta_i/\tau) \geq \exp(-(\delta_x - \varepsilon - \tau \log M)/\tau)\right) \\
&\geq \Pr(\exists\, i, \exp(\delta_i/\tau) \leq \exp((\delta_x - \varepsilon - \tau \log M)/\tau) \\
&= 1 - \prod_{i=1}^{M} \Pr(\delta_i \geq \alpha - \tau \log M) \\
&\geq 1 - \prod_{i=1}^{M} \exp(-\frac{(\alpha_x - \tau \log M)^2 N_i}{2\sigma^2}) \\
&= 1 - \exp(-\frac{(\alpha_x - \tau \log M)^2 n}{2\sigma^2})
\end{aligned}$$

The first inequality comes from our assumption that all $\varepsilon_{i,x} \leq \varepsilon$, and the last inequality comes from the concentration property of subgaussian variables (**Lemma 1**). All other inequalities can be obtained using standard probability arguments. Equation (6) can be derived directly with standard algebra. □

The probability bound provided by **Theorem 2** is much better than the one in **Theorem 1**, since $n$ is the sum of simulation counts of all addressed memory entries, which has to be greater than $n_{min}$.

## Memory-Augmented MCTS

In the previous section, we prove that our memory-based value function approximation is better than the mean outcome evaluation used in MCTS with high probability under mild conditions. The remaining question is to design a practical algorithm and incorporate it with MCTS. In particular, this first requires choosing an approximation of the weight function $\mathbf{w} = f_\tau(-\mathbf{c})$.

### Approximating $\mathbf{w} = f_\tau(-\mathbf{c})$

Let $\phi : \mathcal{S} \to \mathbb{R}^D$ be a function to generate the feature representation of a state. For two states $s, x \in \mathcal{S}$, we approximate the difference between $V^*(s)$ and $V^*(x)$ by a distance function $d(s, x)$ which is set to be the negative cosine of the two states' feature representations:

$$\varepsilon_{s,x} \approx d(s, x) = -\cos(\phi(s), \phi(x)) \tag{7}$$

We apply two steps to create $\phi$. First, take the output of an inner layer of a deep convolutional neural

network and normalize it. We denote this process as $\zeta : \mathcal{S} \rightarrow \mathbb{R}^L$. In practice $L$ will be very large which is time-consuming when computing (7). We overcome this problem by applying a feature hashing function $h : \mathbb{R}^L \rightarrow \mathbb{R}^D$ (Weinberger et al. 2009), and the feature representation is computed by $\phi(s) = h(\zeta(s))$. One nice property of feature hashing is that it can keep the inner product unbiased. Since $\zeta(s)$ is normalized, we have:

$$\mathbb{E}[\cos(\phi(s), \phi(x))] = \cos(\zeta(s), \zeta(x))$$

$\delta_x$ is the term corresponding to the sampling error, which is inversely proportional to the simulation numbers: $\delta_x \propto 1/N_x$. Combining with (7) and the fact that $e^y$ is very close to $y + 1$ for small $y$ we can get our approximation of $f_\tau(-\mathbf{c})$:

$$w_i(x) = \frac{N_i \exp(-d(i, x)/\tau)}{\sum_{j=1}^{M} N_j \exp(-d(j, x)/\tau)} \tag{8}$$

By applying these approximations our model becomes a special case of kernel based methods, such as Locally Weighted Regression and Kernel Regression (Friedman, Hastie, and Tibshirani 2001), where the kernel function can be defined by $k_i(x) = \exp(-d(i, x)/\tau)/\sum_{j=1}^{M} \exp(-d(j, x)/\tau)$. $\tau$ acts like the smoothing factor in those kernel based methods. Our model is also similar to the "attention" scheme used in memory based neural networks (Graves et al. 2016; Weston, Chopra, and Bordes 2015; Vinyals et al. 2016; Pritzel et al. 2017).

## Memory Operations

One memory $\mathcal{M}$ is maintained in our approach. Each entry of $\mathcal{M}$ corresponds to one particular state $s \in \mathcal{S}$. It contains the state's feature representation $\phi(s)$ as well as its simulation statistics $\hat{V}(s)$ and $N(s)$. There are three operations to access $\mathcal{M}$: *update*, *add* and *query*.

**Update** If the simulation statistics of a state $s$ have been updated during MCTS, we also update its corresponding values $\hat{V}(s)$ and $N(s)$ in the memory.

**Add** To include state $s$, we add a new memory entry $\{\phi(s), \hat{V}(s), N(s)\}$. If $s$ has already been stored in the memory, we only update $\hat{V}(s)$ and $N(s)$ at the corresponding entry. If the maximum size of the memory is reached, we replace the least recently *queried* or *updated* memory entry with the new one.

**Query** The *query* operation computes a memory based approximate value given a state $x \in \mathcal{S}$. We first find the top $M$ similar states in $\mathcal{M}$ based on the distance function $d(\cdot, x)$. The approximated memory value is then computed by $\hat{V}_{\mathcal{M}}(x) = \sum_{i=1}^{M} w_i(x)\hat{V}(i)$ where the weights are computed according to equation (8).

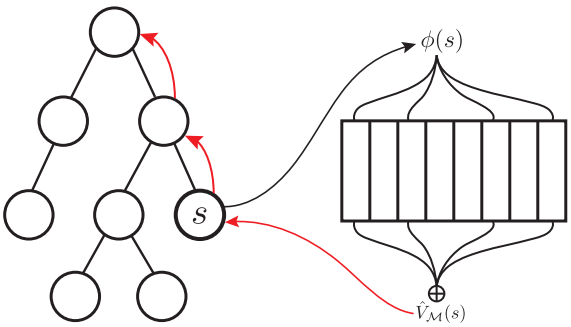



Figure 1: A brief illustration of M-MCTS. When a leaf state $s$ is searched, the feature representation $\phi(s)$ is generated, which is then used to *query* the memory based value approximation $\hat{V}_{\mathcal{M}}(s)$. $\hat{V}_{\mathcal{M}}(s)$ is used to update $s$ and all its ancestors according to equation (9), as indicated by the red arrows in the figure.

The two advantages of addressing the top $M$ similar states are: first, to restrict the maximum value difference of addressed states with $V^*(x)$ within a range, which is shown to be useful in our analysis; second, to make queries in a very large memory scalable. We use an approximate nearest neighbours algorithm to perform the queries based on SimHash (Charikar 2002).

## Integrating Memory with MCTS

We are now ready to introduce our Memory-Augmented MCTS (M-MCTS) algorithm. Figure (1) provides a brief illustration. The main difference between the proposed M-MCTS and regular MCTS is that, in each node of a M-MCTS search tree, we store an extended set of statistics:

$$\{N(s), \hat{V}(s), N_{\mathcal{M}}(s), \hat{V}_{\mathcal{M}}(s)\}$$

Here, $N_{\mathcal{M}}$ is the number of evaluations of the approximated memory value $\hat{V}_{\mathcal{M}}(s)$. During in-tree search of MCTS, instead of $\hat{V}(s)$, we use $(1 - \lambda_s)\hat{V}(s) + \lambda_s\hat{V}_{\mathcal{M}}(s)$ as the value of state $s$, which is used for in-tree selection, for example in the UCB formula. $\lambda_s$ is a decay parameter to guarantee no bias asymptotically.

In original MCTS, a trajectory of visited states $\mathcal{T} = \{s_0, s_1, \ldots, s_T\}$ is obtained at the end of each simulation. The statistics of all states $s \in \mathcal{T}$ in the tree are updated. In M-MCTS, we also update the in-memory statistics by performing the *update*(s) operation of $\mathcal{M}$. Furthermore, when a new state $s$ is searched by MCTS, we compute $\phi(s)$ and use the *add*(s) operation to include $s$ in the memory $\mathcal{M}$.

The most natural way to obtain $\hat{V}_{\mathcal{M}}(s)$ and $N_{\mathcal{M}}(s)$ is to compute and update their value every time $s$ is visited during the in-tree search stage. However, this direct method is time-consuming, especially when the memory size is large. Instead, we only compute the memory

value at the leaf node and backpropagate the value to its ancestors. Specifically, let $s_h \in \mathcal{T}$ be the state just added to the tree whose feature representation $\phi(s_h)$ has already been computed, and its memory approximated value $\hat{V}_{\mathcal{M}}(s_h)$ is computed by *query*$(s_h)$. Let $N_{\mathcal{M}}(s_h) = \sum_{j=1}^{M} k_j(s_h) N_j$, $R = \hat{V}_{\mathcal{M}}(s_h) * N_{\mathcal{M}}(s_h)$. For state $s_i \in \{s_0, \ldots, s_h\}$, we perform the following updates, where $\eta \geq 1$ is a decay parameter.

$$\begin{aligned} X &\leftarrow \max(N_{\mathcal{M}}(s_h)/\eta^{|i-h|}, 1) \\ N_{\mathcal{M}}(s_i) &\leftarrow N_{\mathcal{M}}(s_i) + X \\ \hat{V}_{\mathcal{M}}(s_i) &\leftarrow \hat{V}_{\mathcal{M}}(s_i) + \frac{R - \hat{V}_{\mathcal{M}}(s_i) * X}{N_{\mathcal{M}}(s_i)} \end{aligned} \tag{9}$$

The reason for the decay parameter $\eta$ is because the memory-approximated value of a state is more similar to its closer ancestors.

## Related Work

The idea of utilizing information of similar states has been previously studied in game solver. (Kawano 1996) provided a technique where proofs of similar positions are reused for proving another nodes in a game tree. (Kishimoto and Müller 2004) applied this to provide an efficient Graph History Interaction solution, for solving the game of Checkers and Go.

Memory architectures for neural networks and reinforcement learning have been recently described in Memory Networks (Weston, Chopra, and Bordes 2015), Differentiable Neural Computers (Graves et al. 2016), Matching Network (Vinyals et al. 2016) and Neural Episodic Control (NEC) (Pritzel et al. 2017). The most similar work with our M-MCTS algorithm is NEC, which applies a memory framework to provide action value function approximation in reinforcement learning. The memory architecture and addressing method are similar to ours. In contrast to their work, we provide theoretical analysis about how the memory can affect value estimation. Furthermore, to our best knowledge, this work is the first one to apply a memory architecture in MCTS.

The role of generalization has been previously exploited in transposition tables (Childs, Brodeur, and Kocsis 2008), Temporal-Difference search (TD search) (Silver, Sutton, and Müller 2012), Rapid Action Value Estimation (RAVE) (Gelly and Silver 2011), and mNN-UCT (Srinivasan et al. 2015). A transposition table provides a simple form of generalization. All nodes in the tree corresponding to the same state share the same simulation statistics. Our addressing scheme can closely resemble a transposition table by setting $\tau$ close to zero. In M-MCTS with $\tau > 0$ the memory can provide more generalization, which we show to be beneficial both theoretically and practically.

TD search uses linear function approximation to generalize between related states. This linear function approximation is updated during the online real-time search. However, with complex non-linear function approximation such as neural networks, such updates are impossible to perform online. Since our memory based method is non-parametric, it provides an alternative approach for generalization during real time search.

RAVE uses the all-moves-as-first heuristic based on the intuition that the value of an action is independent of when it is taken. Simulation results are not only updated to one, but to all actions along the simulation path. mNN-UCT applies kernel regression to approximate a state value function, which has been shown equivalent to our addressing scheme using our choice of approximations in Section 4. However, we use the difference between feature representations as the distance metric, while mNN-UCT applies the distance between nodes in the tree. Also, both RAVE and mNN-UCT do not provide any theoretical justifications.

## Experiments

We evaluate M-MCTS in the ancient Chinese game of Go (Müller 2002).

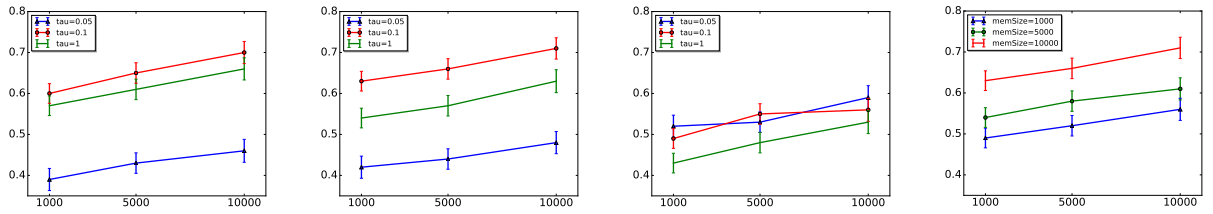
### Implementation Details

Our implementation applies a deep convolutional neural network (DCNN) from (Clark and Storkey 2015), which is trained for move prediction by professional game records. It has 8 layers in total, including one convolutional layer with 64 $7 \times 7$ filters, two convolutional layers with 64 $5 \times 5$ filters, two layers with 48 $5 \times 5$ filters, two layers with 32 $5 \times 5$ filters, and one fully connected layer. The network has about $44\%$ prediction accuracy on professional game records. The feature vector $\phi(s)$ is first extracted from the output of Conv7 which is the last layer before the final fully connected layer of the neural network. The dimension of this output is 23104. A dimension reduction step using feature hashing as described in Section 4 is then applied. The feature hashing dimension is set to 4096, which gives $\phi(s) \in \mathbb{R}^{4096}$.

The hash code in our SimHash implementation has 16 bits. We use 8 hash tables, each of which corresponds to a unique hash function. We also apply a multiple probing strategy. Suppose that a feature vector $\phi(s)$ is mapped to the hash bin $b_i$ at the $i$th hash table. Let the hash code of $b_i$ be $h_i$. To search the neighbours of $\phi(s)$ in the $i$th table, we search those bins whose hash codes' hamming distance to $h_i$ is less than a threshold, set to 1 in our implementation. The discount parameter $\eta$ in equation (9) to update memory approximated values is set to 2.

### Baseline

Our baseline is based on the open source Go program Fuego (Enzenberger and Müller 2008 2017), but adds




(a) $M$=20 (b) $M$=50 (c) $M$=100 (d) Testing Memory Size

Figure 2: Experimental results. Figure (a)-(c) shows the results of testing different value of $M$. Figure (d) shows the results of testing different size of memory. In all figures, x-axis is the number of simulations per move, y-axis means the winning rate against the baseline.

DCNN to improve performance. We adopt the method from (Gelly and Silver 2007) and use DCNN to initialize simulation statistics. Suppose that DCNN is called on the state $s$ that has just been added to the tree. For a move $m$, let $p_m$ be the move probability from the network, and $s'$ the state transformed by taking $m$ on $s$. Let $p_{\max}$ be the maximum of the network's output move probabilities. We compute two statistics $\hat{V}_{\text{DCNN}}(s') = 0.5 * (1.0 + p_m/p_{\max})$ and $\hat{N}_{\text{DCNN}}(s) =$ CNN_STRENGTH $* \, p_m/p_{\max}$. These two values are used as the initial statistics when creating $s'$. We set CNN_STRENGTH to 200 in our experiment.

We implement DCNN in MCTS in a synchronized way, where the search continues after the DCNN evaluation is returned. To increase speed, we restrict DCNN calls to the first 100 nodes visited during the search. This baseline achieves a win rate of 97% against original Fuego with 10,000 simulations per move. We implement M-MCTS based on this baseline. The same DCNN is used to extract features for the memory.

### Results

We first study how the parameters $M$ and $\tau$ can affect the performance of M-MCTS, since these two parameters together control the degree of generalization. The parameter $M$ is chosen from $\{20, 50, 100\}$, and $\tau$ from $\{0.05, 0.1, 1\}$. The size of $\mathcal{M}$ is set to 10000. We vary the number of simulations per move from $\{1000, 5000, 10000\}$. Results are summarized in Figure 2(a)-(c). The best result we have is from the setting $\{M = 50, \tau = 0.1\}$, which achieves a $71\%$ win rate against the baseline with 10,000 simulations per move. The standard error of each result is around $2.5\%$. For $M = 20$ and $M = 50$, the performance of M-MCTS scales well with the number of simulations per move with $\tau = 1$ and $\tau = 0.1$. A small temperature $\tau = 0.05$ cannot beat the baseline at all. We believe the reason is that in this setting M-MCTS only focuses on the closest neighbours for generalization, but does not do enough exploration. For $M = 100$, M-MCTS does not perform well in any setting of $\tau$, since larger $M$ increases the chance of including less similar states.

We then investigate the impact of the size of $\mathcal{M}$ by varying it from $\{1000, 5000, 10000\}$. $M$ and $\tau$ are set to 50 and 0.1 respectively. Results with different number of simulations per move are summarized in Figure 2(d). Intuitively, a large memory can provide better performance, since more candidate states are included for query. The results shown in Figure 2(d) confirm this intuition: M-MCTS achieves the best performance with $|\mathcal{M}| = 10000$, while small memory size $|\mathcal{M}| = 1000$ can even lead to negative effects for value estimation in MCTS. We also compare M-MCTS with the baseline with equal computational time per move. By setting $M = 50$, $\tau = 0.1$ and with 5 seconds per move, M-MCTS achieves a $61\%$ win rate against the baseline.

## Conclusion and Future Work

In this paper, we present an efficient approach to exploit online generalization during real-time search. Our method, Memory-Augmented Monte Carlo Tree Search (M-MCTS), combines the original MCTS algorithm with a memory framework, to provide a memory-based online value approximation. We demonstrate that this can improve the performance of MCTS in both theory and practice. We plan to explore the following two potential future directions. First, we would like to investigate if we can obtain better generalization by combining an offline learned value approximation with our online memory-based framework. Second, the feature representation used in M-MCTS reuses a neural network designed for move prediction. Instead, we plan to explore approaches that incorporate feature representation learning with M-MCTS in an end-to-end fashion, similar to (Pritzel et al. 2017; Graves et al. 2016).

## Acknowledgements

The authors wish to thank Andrew Jacobsen for providing source code of Fuego with the neural network, and the anonymous referees for their valuable advice. This research was supported by NSERC, the Natural Sciences and Engineering Research Council of Canada.

## References


Boucheron, S.; Lugosi, G.; and Massart, P. 2013. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford University Press.

Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 380–388. ACM.

Childs, B. E.; Brodeur, J. H.; and Kocsis, L. 2008. Transpositions and move groups in Monte Carlo tree search. In *IEEE Symposium On Computational Intelligence and Games, 2008.*, 389–395.

Clark, C., and Storkey, A. J. 2015. Training deep convolutional neural networks to play Go. In Bach, F. R., and Blei, D. M., eds., *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Proceedings*, 1766–1774. JMLR.org.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In van den Herik, J.; Ciancarini, P.; and Donkers, H., eds., *Proceedings of the 5th International Conference on Computer and Games*, volume 4630/2007 of *Lecture Notes in Computer Science*, 72–83. Turin, Italy: Springer.

Enzenberger, M., and Müller, M. 2008-2017. Fuego. `http://fuego.sourceforge.net`.

Friedman, J.; Hastie, T.; and Tibshirani, R. 2001. *The elements of statistical learning*, volume 1. Springer series in statistics, Springer, Berlin.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, 273–280. ACM.

Gelly, S., and Silver, D. 2011. Monte-Carlo Tree Search and Rapid Action Value Estimation in computer Go. *Artificial Intelligence* 175(11):1856–1875.

Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwińska, A.; Colmenarejo, S. G.; Grefenstette, E.; Ramalho, T.; Agapiou, J.; et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538(7626):471–476.

Haarnoja, T.; Tang, H.; Abbeel, P.; and Levine, S. 2017. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34nd International Conference on Machine Learning, ICML 2017, Sydney, Australia, 6-11 August 2017*.

Kawano, Y. 1996. Using similar positions to search game trees. In Nowakowski, R. J., ed., *Games of No Chance*, volume 29 of *MSRI Publications*, 193–202. Cambridge University Press.

Kishimoto, A., and Müller, M. 2004. A general solution to the graph history interaction problem. In *Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, 644–649.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 282–293.

Müller, M. 2002. Computer Go. *Artificial Intelligence* 134(1–2):145–179.

Nachum, O.; Norouzi, M.; Xu, K.; and Schuurmans, D. 2017. Bridging the gap between value and policy based reinforcement learning. *arXiv preprint arXiv:1702.08892*.

Pritzel, A.; Uria, B.; Srinivasan, S.; Puigdomènech, A.; Vinyals, O.; Hassabis, D.; Wierstra, D.; and Blundell, C. 2017. Neural episodic control. In *Proceedings of the 34nd International Conference on Machine Learning, ICML 2017, Sydney, Australia, 6-11 August 2017*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Silver, D.; Sutton, R.; and Müller, M. 2012. Temporal-difference search in computer Go. *Machine Learning* 87(2):183–219.

Srinivasan, S.; Talvitie, E.; Bowling, M. H.; and Szepesvári, C. 2015. Improving exploration in UCT using local manifolds. In *AAAI*, 3386–3392.

Tian, Y., and Zhu, Y. 2015. Better computer Go player with neural network and long-term prediction. In *International Conference on Learning Representations*.

Vinyals, O.; Blundell, C.; Lillicrap, T.; Wierstra, D.; et al. 2016. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, 3630–3638.

Weinberger, K.; Dasgupta, A.; Langford, J.; Smola, A.; and Attenberg, J. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 1113–1120. ACM.

Weston, J.; Chopra, S.; and Bordes, A. 2015. Memory networks. In *International Conference on Learning Representations*.

Ziebart, B. D. 2010. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Ph.D.diss., Carnegie Mellon University.